

ASSIGNMENT-03

Name : K. Charan Yadav

HT. No : 2303A52367

Batch: 20

Task 1: Zero-shot Prompt – Fibonacci Series Generator

Scenario : In this task, a zero-shot prompting technique was used. A single comment prompt was written without providing any examples, instructing GitHub Copilot to generate a Python function that prints the first N Fibonacci numbers.

Prompt: # Write a Python function to print the first N Fibonacci numbers

Code:

```
Assignment-3.4.py •
Assignment-3.4.py > ...
1  # Task-1 :| Fibonacci Series Generator
2  def fibonacci(n):
3      """Generate Fibonacci series up to n terms"""
4      a, b = 0, 1
5      for _ in range(n):
6          print(a, end=' ')
7          a, b = b, a + b
8      print()
9
10 # Get input from user
11 num_terms = int(input("Enter number of terms: "))
12 fibonacci(num_terms)
```

Result:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Warning: PowerShell detected that you might be using a screen reader.
able it, run 'Import-Module PSReadLine'.

PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding> & C:/Users/
ve/Desktop/Ai-Assisted Coding/Assignment-3.4.py
Enter number of terms: 7
0 1 1 2 3 5 8
PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding> |
```

Observation:

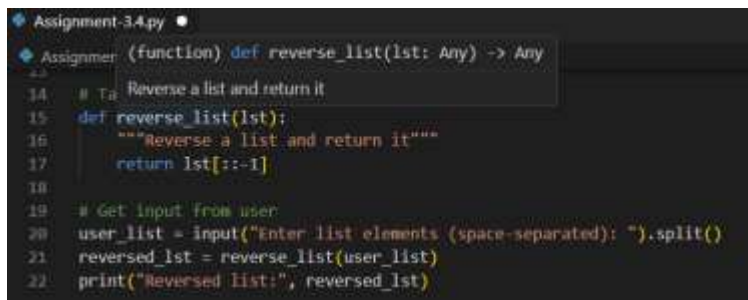
The zero-shot prompt was sufficient for Copilot to correctly infer the Fibonacci logic, even without any examples or additional context. However, the function behavior depended heavily on Copilot's prior training, and the output format was assumed rather than explicitly defined. This shows that zero-shot prompting works well for well-known problems but may lack consistency for ambiguous or complex tasks.

Task 2: One-shot Prompt – List Reversal Function. In this task, a one-shot prompting approach was used by providing a single example along with the instruction to help Copilot generate a correct list reversal function

Prompt: # Write a Python function to reverse a list


Example: input [1, 2, 3] -> output [3, 2, 1]

Code:

A screenshot of a code editor window titled "Assignment-3.4.py". The code defines a function `reverse_list` that takes a list `lst` and returns a new list with elements in reverse order using slicing: `return lst[::-1]`. Below the function, there is a section for user input: `user_list = input("Enter list elements (space-separated): ").split()`, `reversed_list = reverse_list(user_list)`, and `print("Reversed list:", reversed_list)`.

```
Assignment-3.4.py
14 # Task: Reverse a list and return it
15 def reverse_list(lst):
16     """Reverse a list and return it"""
17     return lst[::-1]
18
19 # Get input from user
20 user_list = input("Enter list elements (space-separated): ").split()
21 reversed_list = reverse_list(user_list)
22 print("Reversed list:", reversed_list)
```

Result:

A screenshot of a terminal window showing the execution of the Python script. The prompt is `PS C:\Users\chara\OneDrive\Desktop\AI-Assisted Coding> & C:\Users\chara\OneDrive\Desktop\AI-Assisted Coding\Assignment-3.4.py`. The user enters `1,2,3` when prompted with `Enter list elements (space-separated):`. The output is `Reversed list: ['1', '2', '3']`.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Warning: PowerShell detected that you might be using a screen reader.
Please use the 'Set-PSReadLine' cmdlet to improve the experience.

PS C:\Users\chara\OneDrive\Desktop\AI-Assisted Coding> & C:\Users\chara\OneDrive\Desktop\AI-Assisted Coding\Assignment-3.4.py
Enter list elements (space-separated): 1,2,3
Reversed list: ['1', '2', '3']
PS C:\Users\chara\OneDrive\Desktop\AI-Assisted Coding>
```

Observation:

Providing one example significantly improved Copilot's accuracy and confidence in choosing an optimal approach. The generated solution was concise and efficient, using Python slicing. Compared to zero-shot prompting, one-shot prompting reduced ambiguity and guided Copilot toward the expected output format and logic

Task 3: Few-shot Prompt – String Pattern Matching

Scenario: This task used a few-shot prompting technique by providing multiple examples to help Copilot understand a specific string validation pattern.

Prompt: # Write a function to check if a string starts with a capital letter and ends with a period

Example: "Hello." -> True

Example: "hello." -> False

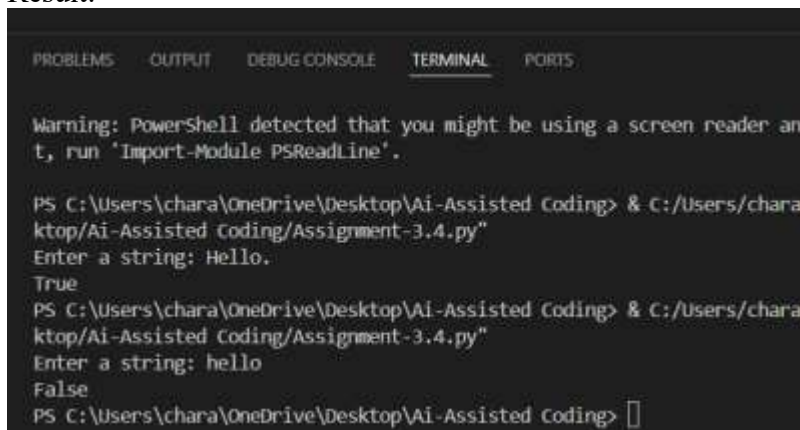
Example: "Hello" -> False

Code:



```
24 # Task-3 : String Pattern Matching
25 def check_pattern(text):
26     # Check if string starts with capital letter and ends with period
27     return text[0].isupper() and text[-1] == '.'
28
29 # Example 1: "Hello." -> True (starts with 'H', ends with '.')
30 # Example 2: "hello." -> False (starts with 'h', not capital)
31 # Example 3: "Hello" -> False (ends with 'o', not period)
32
33 user_text = input("Enter a string: ")
34 if check_pattern(user_text):
35     print("True")
36 else:
37     print("False")
```

Result:



```
Warning: PowerShell detected that you might be using a screen reader and
t, run 'Import-Module PSReadLine'.

PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding> & C:/Users/chara
ktop/Ai-Assisted Coding/Assignment-3.4.py
Enter a string: Hello.
True
PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding> & C:/Users/chara
ktop/Ai-Assisted Coding/Assignment-3.4.py
Enter a string: hello
False
PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding> 
```

Observation:

The few-shot prompt enabled Copilot to accurately identify the string pattern requirements and generate a precise validation function. The multiple examples clarified edge cases and reduced misinterpretation. This demonstrates that few-shot prompting is highly effective when pattern recognition or conditional logic is involved.

Task 4: Zero-shot vs Few-shot – Email Validator

You are participating in a code review session. This task compares zero-shot and few-shot prompting by generating two versions of an email validation function and analyzing their differences

Prompt: Zero-Shot Prompt: # Write a Python function to validate an email address

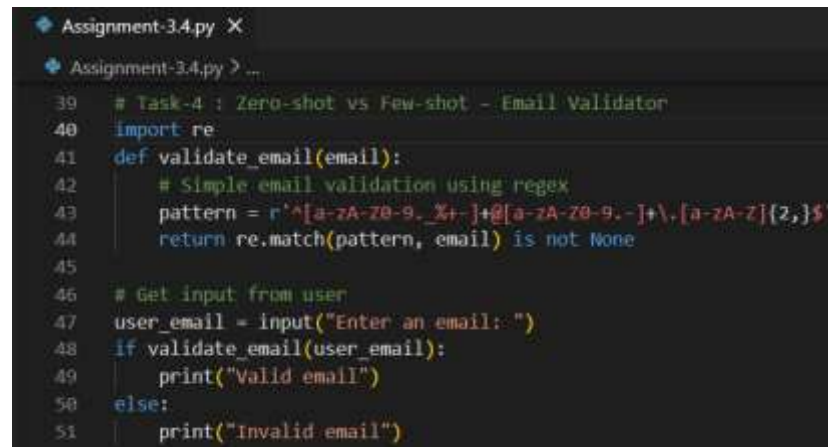
Prompt: Few-Shot Prompt: # Write a Python function to validate an email address

Example: "test@gmail.com" -> True

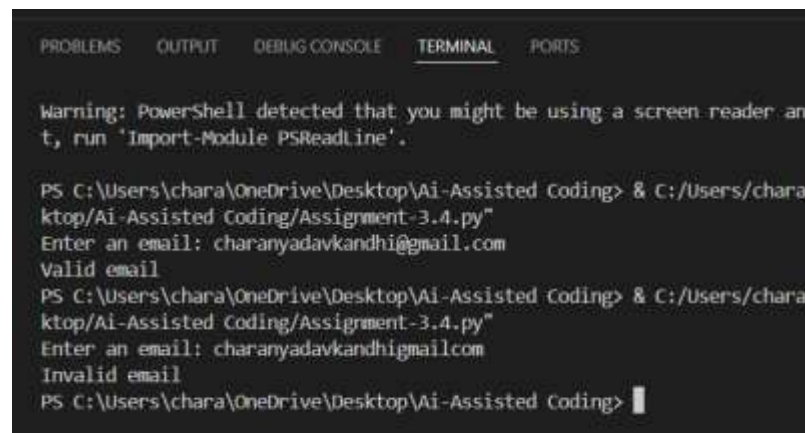
Example: "testgmail.com" -> False

Example: "test@com" -> False

Code:

A screenshot of a code editor window titled 'Assignment-3.4.py'. The code is a Python script for email validation. It starts with a comment '# Task-4 : Zero-shot vs Few-shot - Email Validator', followed by 'import re'. A function 'def validate_email(email):' is defined, containing a comment '# Simple email validation using regex', a regex pattern 'pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}\$'', and a return statement 'return re.match(pattern, email) is not None'. Below the function, there's a comment '# Get input from user', followed by 'user_email = input("Enter an email: ")'. An 'if' statement checks 'if validate_email(user_email):' and prints 'Valid email'. An 'else' block prints 'Invalid email'.

Result:

A screenshot of a PowerShell terminal window. It shows a warning message: 'Warning: PowerShell detected that you might be using a screen reader and, if so, you should run the command "Import-Module PSReadLine".' Below this, the user runs the command 'PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding> & C:/Users/chara/ktop/Ai-Assisted Coding/Assignment-3.4.py'. The terminal then shows the prompt 'Enter an email: charanyadavkandhi@gmail.com' followed by the output 'Valid email'. The user then runs the same command again, but with the email 'charanyadavkandhigmailcom' (missing the dot), and the output is 'Invalid email'. The terminal ends with the prompt 'PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding>'.

Observation:

The zero-shot version produced a very basic and unreliable validation logic, while the few-shot prompt resulted in a more structured and realistic solution. The examples guided Copilot to include domain checks and input validation, significantly improving reliability. This comparison clearly highlights the advantage of few-shot prompting for real-world validation tasks.

Task 5: Prompt Tuning – Summing Digits of a Number. In this task, two different prompt styles were used to study how prompt tuning affects code quality and optimization.

Prompt: Style-1:Generic Prompt # Write a function to return the sum of digits of a number

Prompt with I/O Example: # Write a function to return the sum of digits of a number

Example: sum_of_digits(123) -> 6

Code:

```
Assignment-3.4.py X
Assignment-3.4.py > sum_digits
53 # Task-5 : Summing Digits of a Number
54 def sum_digits(num):
55     # Sum all digits in a number
56     return sum(int(digit) for digit in str(abs(num)))
57
58 # Get input from user
59 user_num = int(input("Enter a number: "))
60 result = sum_digits(user_num)
61 print("Sum of digits:", result)
```

Result:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Warning: PowerShell detected that you might be using a screen reader, run 'Import-Module PSReadLine'.

PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding> & C:/Users/chara/OneDrive/Desktop/Ai-Assisted Coding/Assignment-3.4.py
c:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding\Assignment-3.4.py
pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
Enter a number: 123
Sum of digits: 6
PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding> 
```

Observation:

The prompt that included an input-output example produced a cleaner and more optimized implementation. The example encouraged Copilot to generate concise and Pythonic code using built-in functions. This demonstrates that prompt tuning with examples not only improves correctness but also enhances code quality and efficiency.